BUGFYND · APPLICATION SECURITY

Thick Client Application Penetration Testing Report

Prepared for: Sample Client Pvt Ltd
Application: BetaBank (two-tier .NET banking client)

Version 1.1 · Final report

SAMPLE REPORT: findings come from a deliberately vulnerable lab application, not from a real client

Confidentiality statement

This document is confidential and intended only for the client named on the cover. It may not be copied, shared or published, in whole or in part, without written permission from the client and Bugfynd. The client may share it with its own auditors, customers and partners to support its security and compliance programs.

About this sample: this report shows the format and level of detail Bugfynd delivers after a thick client penetration test. All findings come from BetaBank, an open-source, intentionally vulnerable two-tier .NET banking application (NetSPI thick client lab) tested in our lab. Client, dates and evidence are illustrative; secrets are redacted.

Contents

#	Section
1	Document details
2	Introduction, scope and duration
3	Executive summary and risk ratings
4	Approach and methodology
5	Results overview and high-level recommendations
6	Detailed findings (6)
7	Conclusion
A–C	Annexures: changes to environment, tools used, tests performed

SAMPLE

1. Document details

Document control

Document title	Thick Client Penetration Testing Report
Classification	Confidential – Final report
Report version	1.1 (includes retest results)

Document history

Version	Date	Prepared by	Status
1.0	[Date]	Bugfynd testing team	Initial report
1.1	[Date]	Bugfynd testing team	Final report after retest

Client information

Company	Sample Client Pvt Ltd
Point of contact	[Name, title]
Email	[contact@client.example]

Bugfynd team

Name	Role	Responsibility
Consultant A	Lead Consultant	Testing and report preparation
Consultant B	Security Consultant	Technical review
Consultant C	Security Consultant	Testing support and retest
[Client contact]	Client representative	Report acceptance

SAMPLE

2. Introduction

Bugfynd performed a manual penetration test of the client's two-tier Windows banking application, where the desktop client connects directly to a Microsoft SQL Server database to find security weaknesses an attacker could exploit, measure their business impact and give clear steps to fix them. Testing was carried out under written authorization from the client and within the agreed rules of engagement.

Scope

Application	Type	Version	Components
BetaBank	.NET WPF Windows desktop (two-tier)	[x.y]	BetaBank.exe, configuration files, registry keys, Microsoft SQL Server backend

Duration

Activity	Dates
Initial assessment including reporting	[Start date] – [End date]
Retest and report update	[Retest date]

Test approach

- Grey-box testing with standard user and administrator test accounts provided by the client.
- Testing performed from Bugfynd's lab environment on a dedicated Windows test machine.
- No denial-of-service or destructive testing was performed.

Limitations

- Results reflect the application version and environment available during the test window.
- Server-side components were tested only where reachable from the client application.

SAMPLE

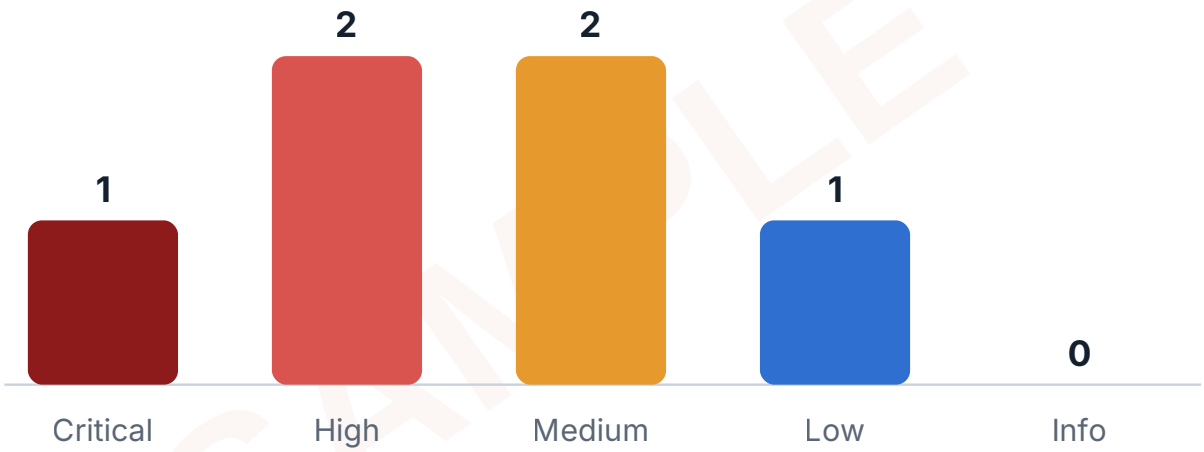
3. Executive summary

Bugfynd identified **6 security findings** in the application: **1 Critical, 2 High, 2 Medium and 1 Low**. The most serious issue lets any user transfer money from other customers' accounts, because authorization checks run only in the desktop client. The client also connects straight to the database with stored credentials, so a user who extracts them can read and change every account.

After the client's fixes, a retest confirmed that **5 of 6** findings are remediated. One High finding (direct database access from the client) needs an architecture change and remains open with an agreed plan.

1	2	2	1	0
Critical	High	Medium	Low	Info

Findings by severity



Initial test vs retest

Status	Critical	High	Medium	Low	Total
Initial assessment	1	2	2	1	6
Open after retest	0	1	0	0	1

SAMPLE

Risk rating

Each finding is scored with **CVSS v3.1** (Common Vulnerability Scoring System), the industry standard for rating how severe a vulnerability is. Scores run from 0 to 10.

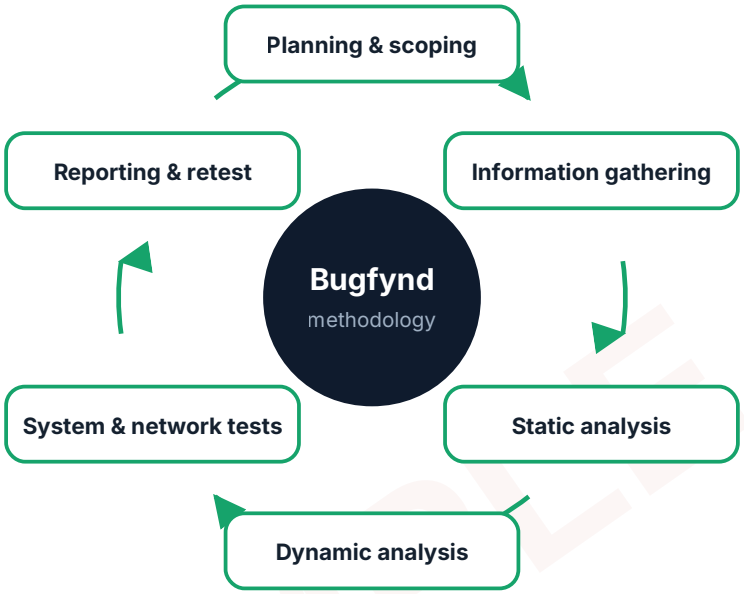
Severity	CVSS score	What it usually means
Critical	9.0 – 10.0	Easy to exploit, often remotely and without credentials; can fully compromise the application or its data. Fix immediately.
High	7.0 – 8.9	Exploitable with little effort; gives access to sensitive data or higher privileges. Fix as a priority.
Medium	4.0 – 6.9	Needs specific conditions or some access; limited impact on its own but can be chained. Fix in the next release.
Low	0.1 – 3.9	Minor impact or hard to exploit; mainly defence-in-depth. Fix as part of normal hardening.
Info	0.0	Best-practice observation with no direct security impact.

Scores are calculated with the FIRST CVSS 3.1 calculator (first.org/cvss/calculator/3.1) and adjusted for the client's environment where relevant.

SAMPLE

4. Approach and methodology

Bugfynd's thick client methodology combines manual testing with tool support and follows recognised standards: **OWASP Desktop App Security Top 10**, **OWASP Top 10**, **OWASP API Security Top 10**, **CWE Top 25** and the **Penetration Testing Execution Standard (PTES)**.



Phase	What we do
Planning & scoping	Agree scope, test accounts, testing windows and contacts; obtain written authorization.
Information gathering	Map the architecture, technologies, entry points, files, registry use, network endpoints and dependencies.
Static analysis	Decompile binaries (dnSpy, ILSpy), review configuration files and stored data, look for hard-coded secrets and weak cryptography, check binary protections.
Dynamic analysis	Test inputs for injection, authentication and authorization flaws, GUI control bypass, file handling and business logic while the application runs.
System & network tests	Inspect memory, file and registry permissions, DLL loading and traffic (TLS, cleartext protocols, request tampering).
Reporting & retest	Document every finding with impact, steps, evidence and fixes; report critical issues the same day; retest after fixes.

Communication

- Daily status updates during testing.
- Critical findings reported to the client contact the same day.
- Walkthrough call with the development team after report delivery.

SAMPLE

5. Results overview

#	Finding	Severity	CVSS	Retest status
1	Client-side Authorization Bypass in Fund Transfer	Critical	9.1	Remediated
2	SQL Injection in Account Search	High	8.1	Remediated
3	Direct Database Access with Stored Credentials	High	8.8	Open
4	Hidden Administrator Functions Enabled via GUI Manipulation	Medium	6.5	Remediated
5	Weak Password Storage (Unsalted MD5)	Medium	5.9	Remediated
6	Sensitive Data Stored in Registry	Low	3.3	Remediated

High-level recommendations

1. Enforce every authorization decision on the server, never in the desktop client.
2. Move from two-tier to three-tier: the client should talk to an authenticated API, not directly to the database.
3. Remove database credentials and secrets from the client, configuration and registry.
4. Use parameterized queries for all database calls.
5. Hash passwords with a modern algorithm (bcrypt, scrypt or Argon2) and enforce account lockout.

SAMPLE

6.1 Client-side Authorization Bypass in Fund Transfer

Critical

Severity / CVSS	Critical · 9.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`
Affected component	BetaBank.exe – Transfer screen
Category	CWE-602
Retest status	Remediated
Description	The client checks that the source account belongs to the logged-in user only inside the desktop application. The database accepts any account number sent by the client, so a modified client or tampered request can move money from any account.
Impact	A logged-in customer can transfer funds out of other customers' accounts, causing direct financial loss and fraud.
Steps to reproduce	1. Log in as test user A and open the Transfer screen. 2. Use dnSpy to set a breakpoint in the transfer method and change the source account number to test user B's account. 3. Continue execution: the transfer completes and B's balance decreases.

Evidence	`Source account (UI):  [User A account]`
	`Source account (patched): [User B account]`
	`Result: Transfer successful — User B balance reduced`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Perform all authorization checks on the server for every transaction. • Bind the source account to the authenticated session on the server. • Log and alert on transfers from accounts not owned by the requester.
Reference	https://cwe.mitre.org/data/definitions/602.html

SAMPLE

6.2 SQL Injection in Account Search

High

Severity / CVSS	High · 8.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`
Affected component	BetaBank.exe – Account search
Category	CWE-89
Retest status	Remediated
Description	The account search box builds a SQL statement by joining strings. Special characters are not escaped, so input can change the query sent to the database.
Impact	An attacker can read data from any table, including other customers' details and balances, and may modify records.
Steps to reproduce	1. Log in as a normal user. 2. Enter `' UNION SELECT name, NULL, NULL FROM sys.tables --` in the search box. 3. Table names from the database appear in the results grid.

Evidence	`Search input : ' UNION SELECT name,NULL,NULL FROM sys.tables --`
	`Results grid : Accounts, Users, Transactions, ...`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Use parameterized queries for every database call. • Run the application's database account with least privilege. • Validate search input length and characters.
Reference	https://owasp.org/www-community/attacks/SQL_Injection

SAMPLE

6.3 Direct Database Access with Stored Credentials

High

Severity / CVSS	High · 8.8
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H`
Affected component	BetaBank.exe.config / two-tier design
Category	CWE-798
Retest status	Open
Description	The client connects directly to SQL Server using a shared database account. The connection string is stored in the configuration file and can be recovered by any user of the machine.
Impact	Anyone who extracts the connection string can connect to the database with a SQL client and bypass the application entirely, reading or changing every account.
Steps to reproduce	1. Open BetaBank.exe.config in the installation folder. 2. Copy the connection string (server, database, user and password). 3. Connect with SQL Server Management Studio and query the Accounts table.

Evidence	`connectionString="Server=[db-host];Database=BetaBank;User Id=[REDACTED];Password=[REDACTED]"`
	`SSMS login: successful — full access to Accounts table`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Introduce a server-side API layer (three-tier) and remove database access from the client. • Until then, use Windows Integrated Authentication with per-user, least-privilege database roles. • Rotate the exposed database password.
Reference	https://cwe.mitre.org/data/definitions/798.html

SAMPLE

6.4 Hidden Administrator Functions Enabled via GUI Manipulation

Medium

Severity / CVSS	Medium · 6.5
CVSS vector	`CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`
Affected component	BetaBank.exe – Main window
Category	CWE-284
Retest status	Remediated
Description	Administrator features such as 'View all customers' are present in the normal user's window but hidden or disabled. They can be enabled with a window inspection tool and then used.
Impact	A normal user can view other customers' personal details by using admin-only screens.
Steps to reproduce	1. Log in as a normal user. 2. Open WinSpy++ and select the main BetaBank window. 3. Set the hidden 'Admin' button to visible and enabled, then click it: the customer list opens.

Evidence	`Control: btnAdminPanel  Visible: False -> True  Enabled: False -> True`
	`Result: 'All customers' screen opened as normal user`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Do not ship admin features in the standard client, or load them only after a server-side role check. • Check the user's role on the server for every admin action.
Reference	https://cwe.mitre.org/data/definitions/284.html

SAMPLE

6.5 Weak Password Storage (Unsalted MD5)

Medium

Severity / CVSS	Medium · 5.9
CVSS vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N`
Affected component	Users table / login routine
Category	CWE-916
Retest status	Remediated
Description	User passwords are stored as unsalted MD5 hashes. MD5 is fast and has no salt, so most passwords can be cracked quickly with public wordlists.
Impact	If the database is exposed (see findings 2 and 3), attackers can recover many users' plain-text passwords and reuse them elsewhere.
Steps to reproduce	1. Decompile BetaBank.exe and review the login method. 2. Observe that the password is hashed with MD5 before comparison. 3. Crack a sample hash from a test account with a wordlist in under a minute.

Evidence	`Login(): hash = MD5(password)  // no salt`
	`Test hash cracked with wordlist: yes (under 60 seconds)`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Store passwords with bcrypt, scrypt or Argon2 using a unique salt. • Force a password reset for all users after migration. • Perform password checks on the server, not in the client.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

SAMPLE

6.6 Sensitive Data Stored in Registry

Low

Severity / CVSS	Low · 3.3
CVSS vector	`CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N`
Affected component	HKCU\Software\BetaBank
Category	CWE-312
Retest status	Remediated
Description	The client saves the last logged-in username and account number in the registry in clear text.
Impact	Another user or malware on the same machine can learn customer identifiers, which helps targeted attacks.
Steps to reproduce	1. Take a Regshot snapshot, log in to BetaBank, then take a second snapshot. 2. Compare the snapshots and open the new BetaBank key.

Evidence	`HKCU\Software\BetaBank\LastUser = [username]`
	`HKCU\Software\BetaBank\LastAccount = [account number]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Do not store customer identifiers locally; if needed, protect them with Windows DPAPI. • Clear stored values on logout.
Reference	https://cwe.mitre.org/data/definitions/312.html

7. Conclusion

The application relied on the desktop client to enforce security, which an attacker fully controls. The most urgent issues, client-side authorization bypass and SQL injection, have been fixed and verified. Bugfynd recommends completing the move to a three-tier design so that the database is no longer reachable from user machines, then retesting.

Annexure A: Changes to environment

No permanent changes were made. Test transfers were made only between test accounts provided by the client and were reversed with the client's help after testing.

Annexure B: Tools used

Tool	Purpose
dnSpy / ILSpy	Decompiling and debugging .NET binaries
Sysinternals (Procmon, Process Explorer, AccessChk)	Process, file, registry and permission analysis
Process Hacker	Memory inspection and process dumps
Wireshark	Network traffic capture and analysis
Echo Mirage / Burp Suite	Intercepting and modifying application traffic
PESecurity	Checking binary protections (ASLR, DEP, CFG)
Regshot	Comparing registry changes
sslscan	TLS configuration review

SAMPLE

Annexure C: Tests performed

Test area	Status
Information gathering and architecture review	Tested
Authentication and authorization	Tested
Input validation and injection	Tested
GUI control and object permissions	Tested
File and configuration storage	Tested
Registry storage and permissions	Tested
Hard-coded secrets and cryptography	Tested
Memory analysis	Tested
DLL loading and binary protections	Tested
Network communication and TLS	Tested
Traffic interception and tampering	Tested
Error handling and information disclosure	Tested
Business logic	Tested

SAMPLE